

Programming Engineering

Course 2 – 24 February
adiftene@info.uaic.ro

Content

- ▶ From Course 1 ...
- ▶ Prototyping
- ▶ RUP
- ▶ V-Model
- ▶ Extreme Programming
- ▶ Agile
- ▶ Scrum
- ▶ Lean, Kanban
- ▶ MDD, AMDD
- ▶ TDD
- ▶ Requirement analysis

From Course 1 – 1

- ▶ Programming Engineering (Software engineering)
 - Methodologies used to develop large software projects by teams
 - *Establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines*
 - Classical phases: *analysis, design, developing, testing, deployment, maintenance, withdrawal of software products*
 - Other phases: *budget and resources administration, team coordination, planning*
- ▶ **Aim:** to obtain safe programs that operate efficiently on concrete computing machines

From Course 1 – Phases

- ▶ Requirements engineering
- ▶ Architectural design
- ▶ Detailed design
- ▶ Implementation
- ▶ Integration
- ▶ Validation
- ▶ Verification
- ▶ Deployment
- ▶ Maintenance

Prototyping

► Types of prototypes

◦ Throw-away

- Aim: clarify specifications
- It grows quickly, everything else is secondary
- Useful for solving “architectural/technological spikes”
- The "true" program is written then from 0

◦ **Evolutionary**

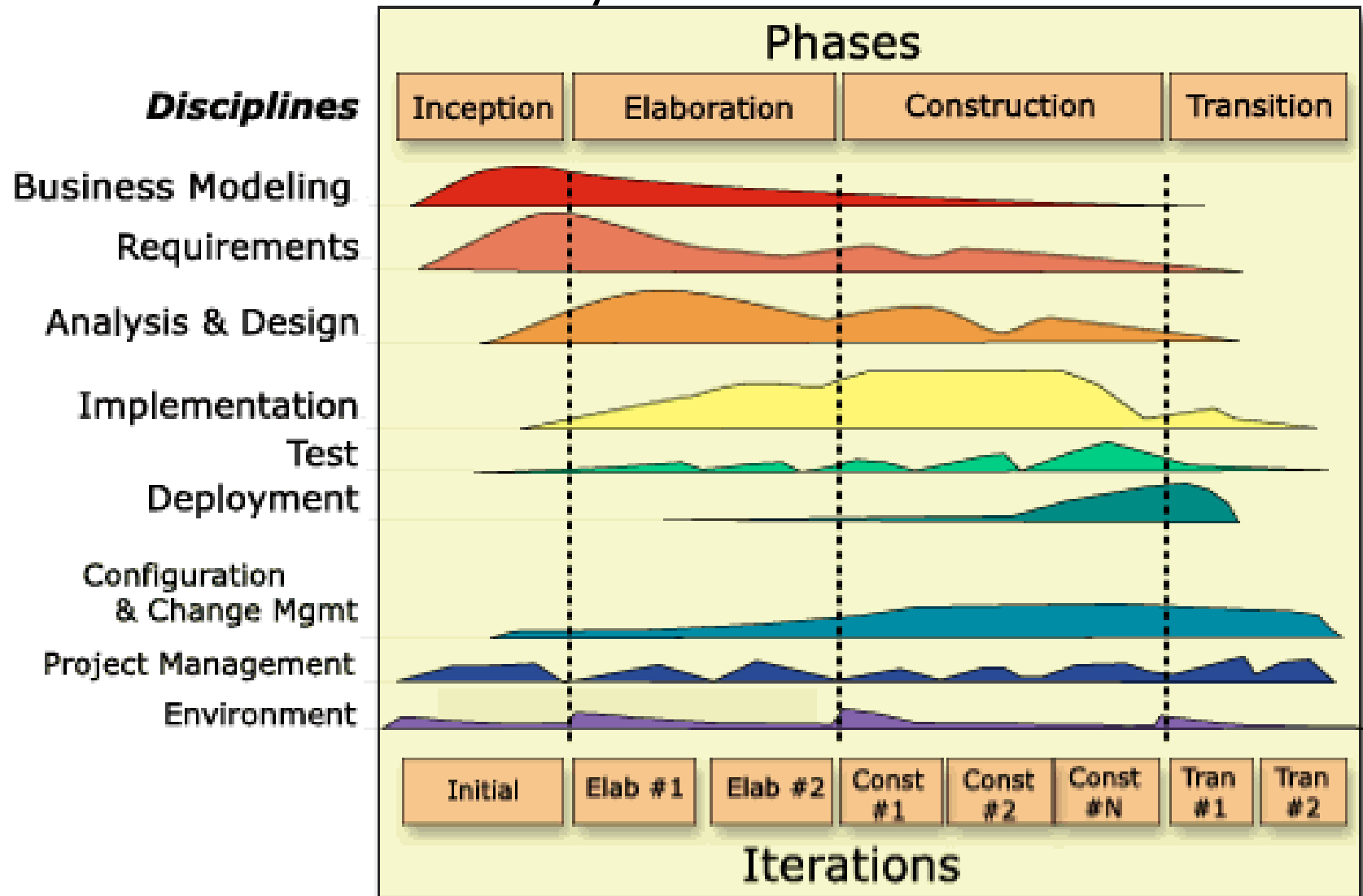
- Aim: incremental building of final product
- It builds a functional core with further functionalities added

Prototyping +-

- ▶ +: It can remove unclear specifications
- ▶ +: Customers can change the requirements (it is cheaper to manage)
- ▶ +: Inexpensive maintenance (continuous assessment)
- ▶ +: It can facilitate user training
- ▶ -: Artificial environment, hidden problems
- ▶ -: *Why is it almost ready ?! Why is it taking so long?*
- ▶ -: *May we change specifications? Well I want and ...*
- ▶ -: *Do you mean my work is thrown away?*

RUP (Rational Unified Process)

- ▶ Iterative model used by IBM from 2003

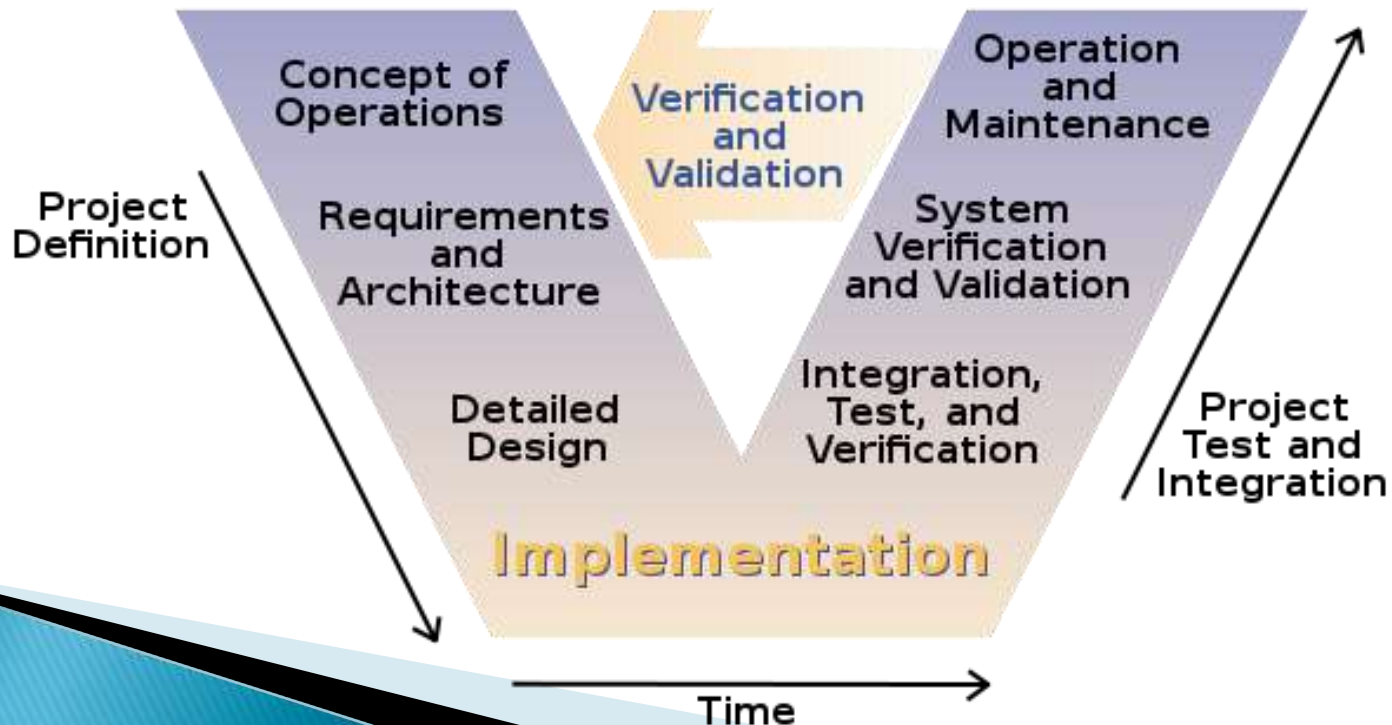


RUP

- ▶ The engineering of functionality. Functional needs are synthesized.
- ▶ Four project life-cycle phases:
 - **Inception:** basis for validating initial costs and budgets, initial risk assessment and project description
 - **Elaboration:** problem domain analysis, the architecture of the project gets its basic form
 - **Construction:** the development of components and other features of the system
 - **Transition:** the transition of the system from development towards production

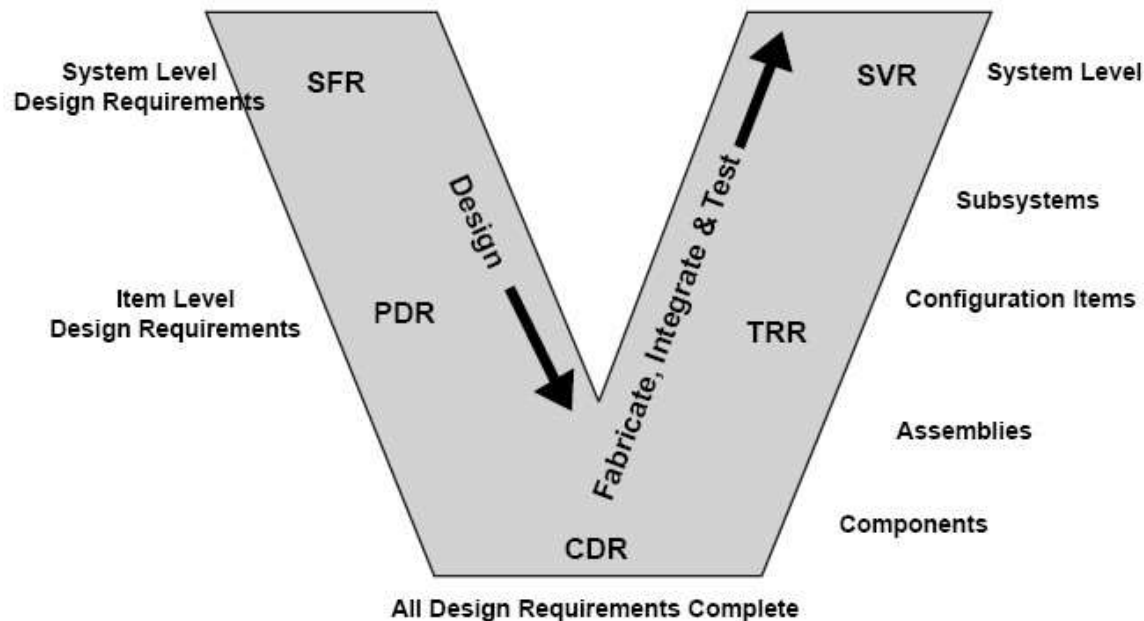
V-Model

- ▶ Used in Germany and US in '80
- ▶ **Left side** – requirements analysis and development of design
- ▶ **Right side** – integration and validation
- ▶ V from Validation and Verification



V-Model – Objectives

- ▶ Minimizing risk
- ▶ Improvement and quality assurance
- ▶ Cost reduction
- ▶ Improving communication



SFR = System Functional Review
PDR = Preliminary Design Review
CDR = Critical Design Review

TRR = Test Readiness Review
SVR = System Verification Review

V-Model + -

- ▶ +: V-model users participate both in development and maintenance
- ▶ +: There is a group that controls changes in specifications. It meets once a year and decides what changes are accepted
- ▶ +: The model provides assistance at each stage and explicitly defines what to do
- ▶ -: It doesn't ensure maintenance
- ▶ -: It is used to relatively small projects

Extreme Programming

► Why "Extreme"?

- "Extreme" means these practices get "turned up" to a much higher "volume" than the one of traditional projects.

► What really matters?

- Listening, Testing, Coding, Designing



XP – Characteristics

- ▶ XP is a modern, lightweight developing model **based on RUP**
- ▶ Program development does not mean hierarchies, responsibilities and deadlines, but **collaboration** of people that make up the **team**
- ▶ Team members are encouraged to **assert their personality**, to give and receive knowledge and become **brilliant programmers**
- ▶ XP believes that developing programs primarily means **writing programs** (PowerPoint files can not be compiled)

Main ideas in XP

- ▶ The project is **in the minds of all programmers** inside the team, not in documentation, models or reports
- ▶ At any time, a **customer representative** is available to clarify requirements
- ▶ Code is written as simple as possible. **Test code is written first**
- ▶ If there is a need for rewriting or for **throwing away code**, it is done with no mercy
- ▶ **Integrated code changes continuously** (several times per day)
- ▶ It is programmed in teams (**pair programming**). The teams changed at the end of iterations (1–2 weeks)
- ▶ It implies up to **40 hours per week** without additional work

XP Rules

Planning

- ❖ 2 User stories are written.
- ❖ 2 Release planning creates the schedule.
- ❖ 2 Make frequent small releases.
- ❖ 2 The Project Velocity is measured.
- ❖ 2 The project is divided into iterations.
- ❖ 2 Iteration planning starts each iteration.
- ❖ 2 Move people around.
- ❖ 2 A stand-up meeting starts each day.
- ❖ 2 Fix XP when it breaks.

Designing

- ❖ 2 Simplicity.
- ❖ 2 Choose a system metaphor.
- ❖ 2 Use CRC cards for design sessions.
- ❖ 2 Create spike solutions to reduce risk.
- ❖ 2 No functionality is added early.
- ❖ 2 Refactor whenever and wherever possible.

Coding

- ❖ 2 The customer is always available.
- ❖ 2 Code must be written to agreed standards.
- ❖ 2 Code the unit test first.
- ❖ 2 All production code is pair programmed.
- ❖ 2 Only one pair integrates code at a time.
- ❖ 2 Integrate often.
- ❖ 2 Use collective code ownership.
- ❖ 2 Leave optimization till last.
- ❖ 2 No overtime.

Testing

- ❖ 2 All code must have unit tests.
- ❖ 2 All code must pass all unit tests before it can be released.
- ❖ 2 When a bug is found tests are created.
- ❖ 2 Acceptance tests are run often and the score is published.

Agile

- ▶ Rapid customer satisfaction by providing **continuous useful software** (weekly if possible)
- ▶ The progress is measured in terms of the **functional part of the project**
- ▶ **Even late changes** in requirements are welcome
- ▶ A **very close cooperation** between the customer and programmers
- ▶ The **talks face-to-face** is the best form of communication
- ▶ **Continuous adaptation** to changes occurring
- ▶ Develop a spirit of highlighting and solving problems, not hiding or failing to observe them

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck

Mike Beedle

Arie van Bennekum

Alistair Cockburn

Ward Cunningham

Martin Fowler

James Grenning

Jim Highsmith

Andrew Hunt

Ron Jeffries

Jon Kern

Brian Marick

Robert C. Martin

Steve Mellor

Ken Schwaber

Jeff Sutherland

Dave Thomas

Agile +-

▶ -:

- Failure to achieve the **necessary documentation**
- It only works with “**senior-level**” developers
- Insufficient structuring of software modeling
- **Contract negotiations may be difficult**

▶ +:

- Companies that have adopted “**Toyota way of working**” improved their productivity by 83%, with 93% time of production, with 91% product quality, and they have halved the overtime – according to U.S. official study conducted a few years ago on companies in the automotive industry

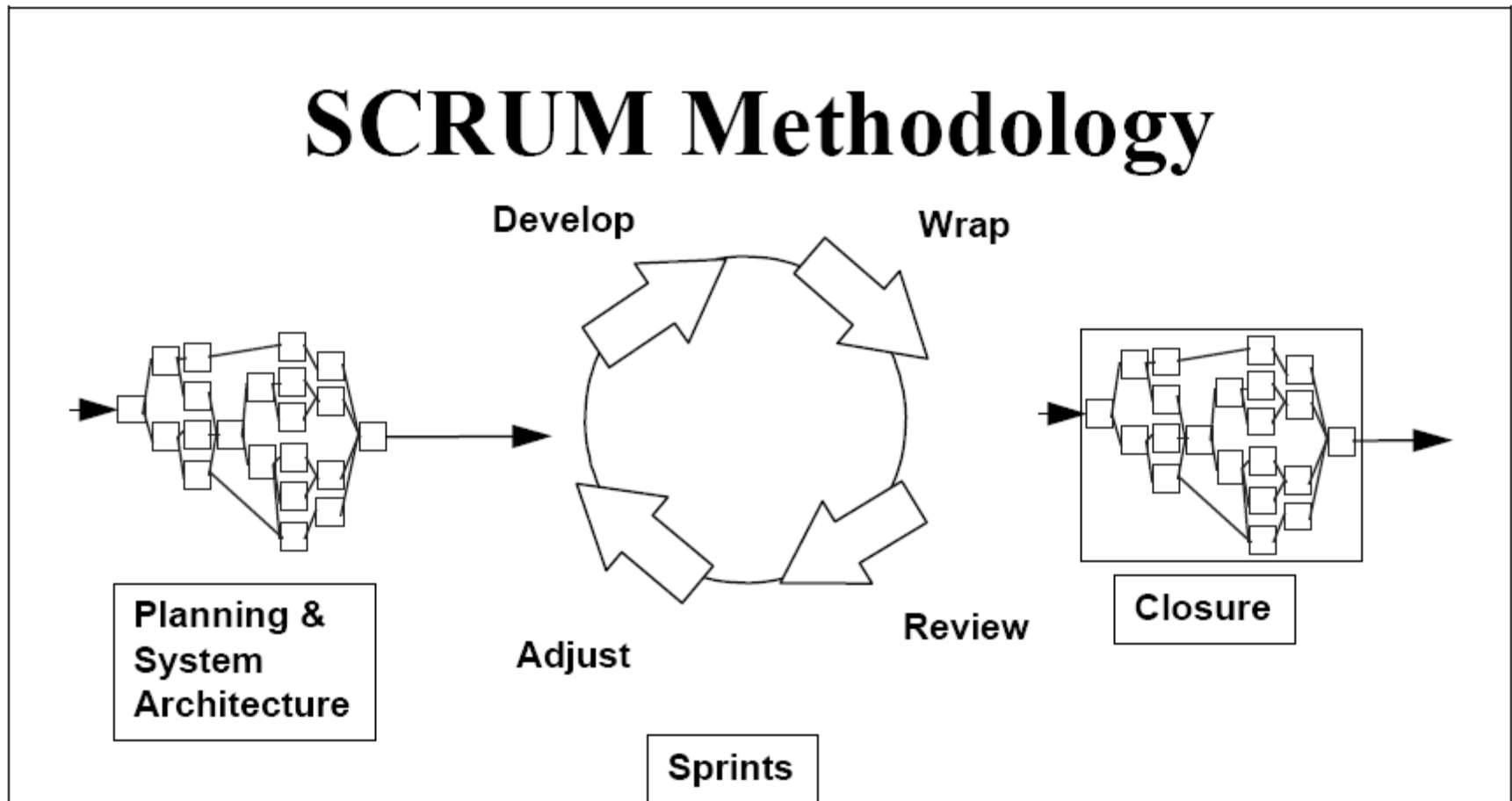


Scrum



- ▶ The customer becomes part of the development team
- ▶ Common intermediate distributions of the software, with immediate checks and validations
- ▶ Daily Discussions:
 - What did you do yesterday? (Achievements)
 - What are you going to do tomorrow? (To achieve)
 - What are the problems that you may become tangled? (Problems / risks)
- ▶ Transparency in planning and development
- ▶ Frequent meetings to monitor progress
- ▶ No problems kept under carpet
- ▶ Work efficiency: "working more hours" does not mean "more results"

Scrum Methodology



Lean

- ▶ Lean Software Development (LSD) Principles:
 1. **Value** – identify what is really important to the customer and focus on that
 2. **Value Stream** – ensure all activities are necessary and add value
 3. **Flow** – strive for continuous processing through the value stream
 4. **Pull** – drive production with demand
 5. **Perfection** – prevent defects and redundant work

Kanban

- ▶ A scheduling system for lean and **just-in-time** (JIT) production
- ▶ Taiichi Ohno at Toyota in 1953
- ▶ A system to control the logistical chain from the production point of view, and is an **inventory** control system
- ▶ Aligns inventory levels with actual consumption. A signal tells a supplier to produce and deliver a new shipment when material is consumed
- ▶ An approach where the "pull" comes from demand

Kanban – Toyota's Six Rules

- ▶ Later process picks up the number of items indicated by the Kanban at the earlier process.
- ▶ Earlier process produces items in the quantity and sequence indicated by the Kanban.
- ▶ No items are made or transported without a Kanban.
- ▶ Always attach a Kanban to the goods.
- ▶ Defective products are not sent on to the subsequent process. The result is 100% defect-free goods.
- ▶ Reducing the number of Kanban increases the sensitivity.

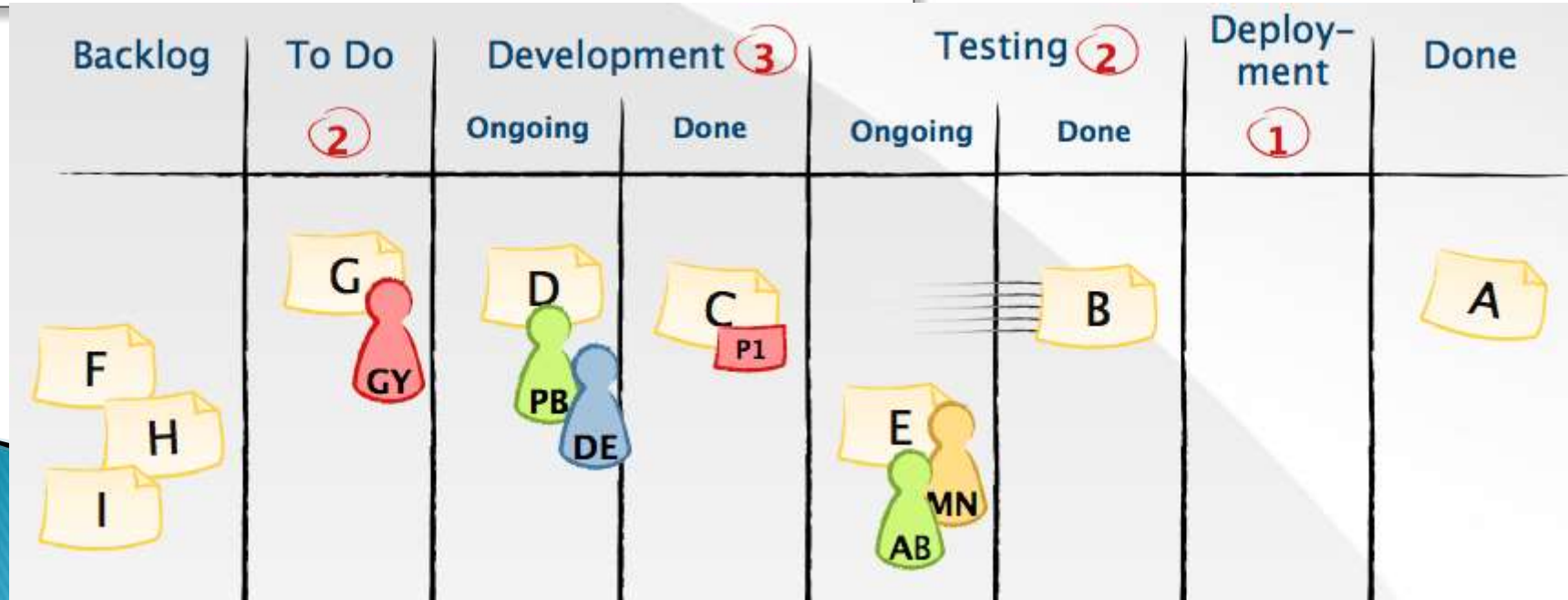
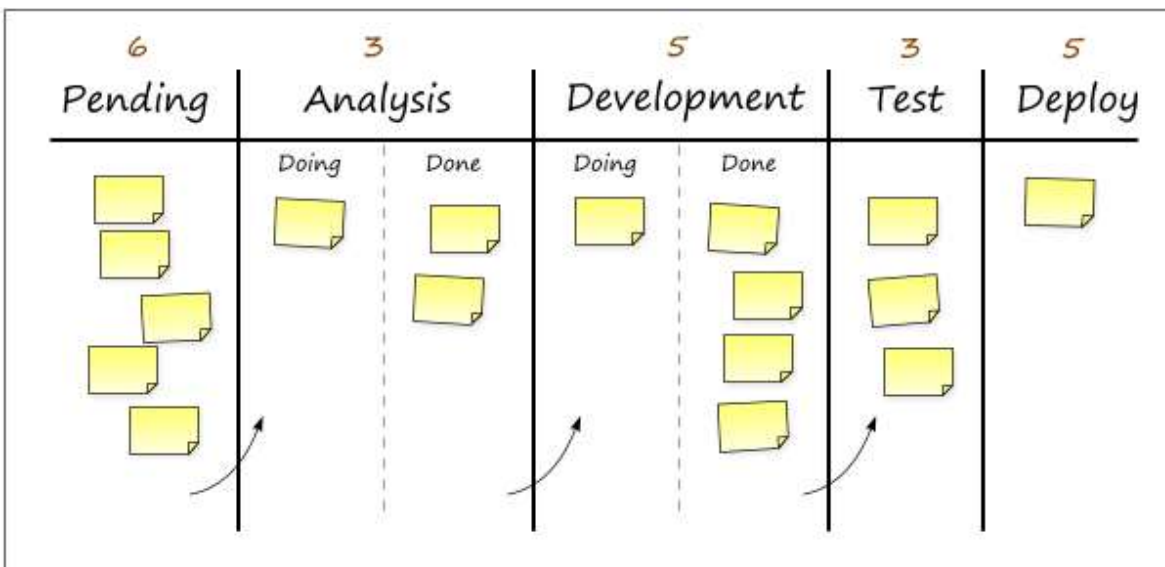
Kanban methodology

- ▶ 2009, Corey Ladas, Scrumban, 2010, David Anderson
- ▶ A method for managing knowledge work with an emphasis on **just-in-time delivery** while not **overloading the team members**
- ▶ A visual process-management system that tells what to produce, when to produce it, and how much to produce
- ▶ Visualisation is an important aspect that allows to understand the work and the workflow

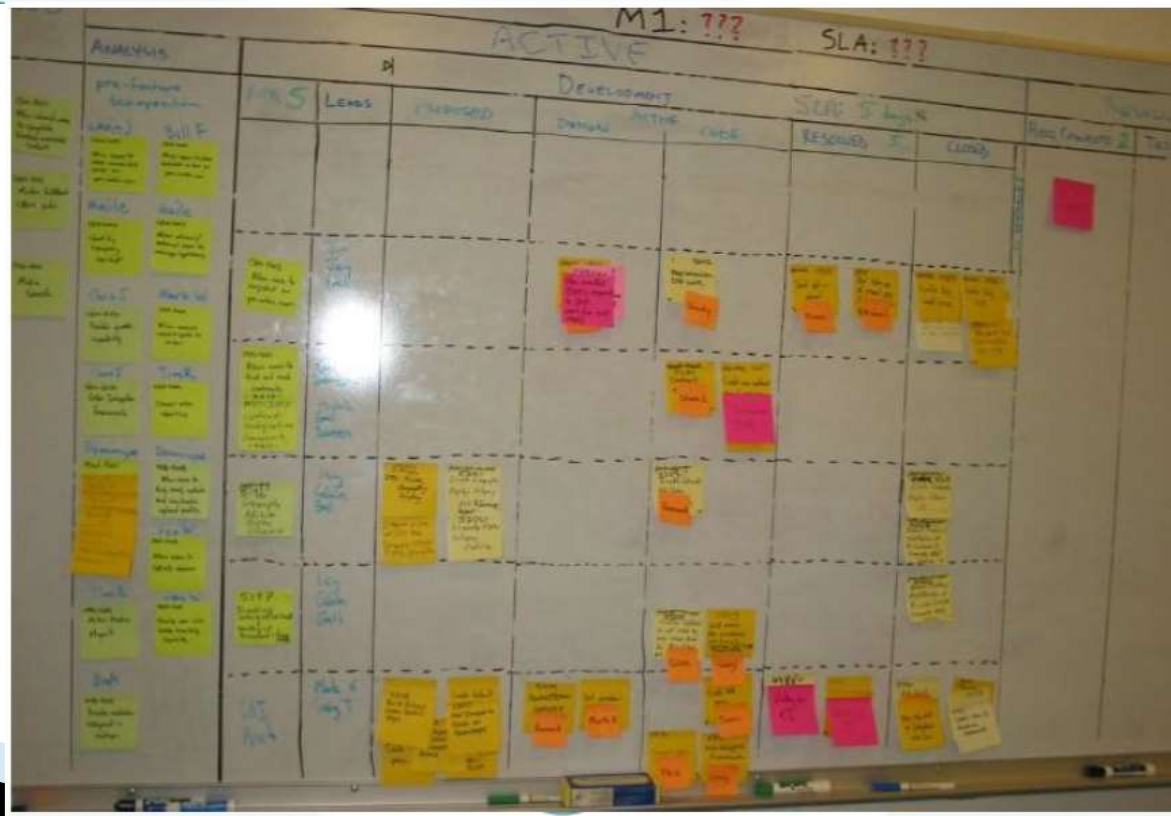
Kanban principles

- ▶ **Start with existing process** – The Kanban method does not prescribe a specific set of roles or process steps
- ▶ **Agree to pursue incremental, evolutionary change** – continuous, incremental and evolutionary change is the way to make system improvements and make them stick
- ▶ **Respect the current process, roles, responsibilities and titles** – agreeing to respect current roles, responsibilities and job titles with the goal of gaining further support
- ▶ **Leadership at all levels** – Acts of leadership at all levels in the organization are encouraged

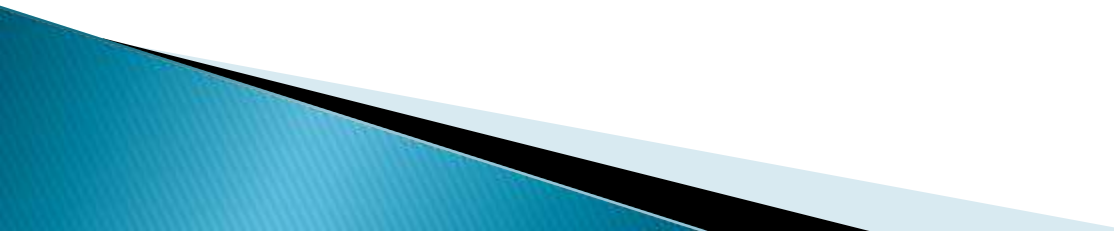
Kanban – Visual Example



Kanban – Examples

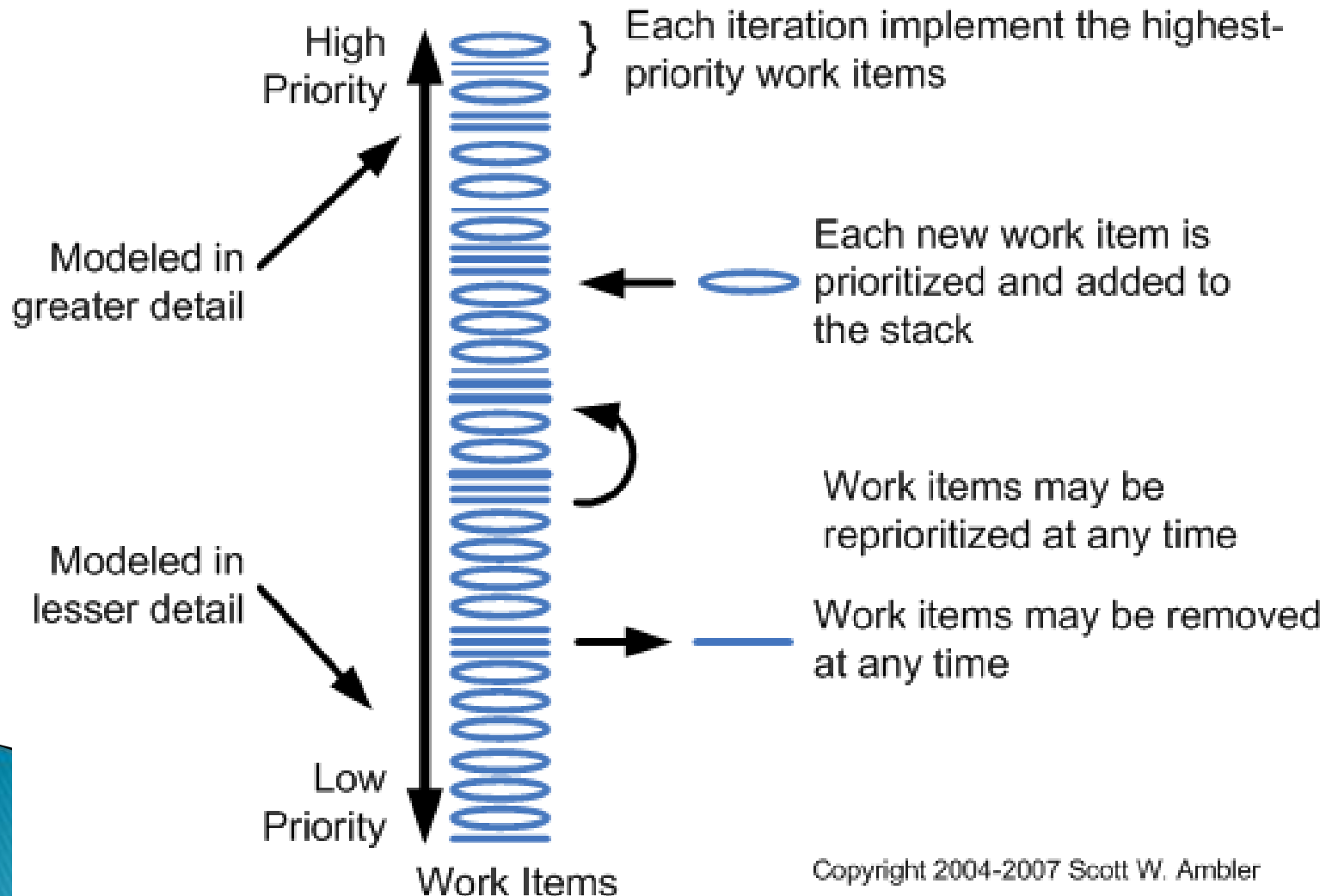


Model Driven Development

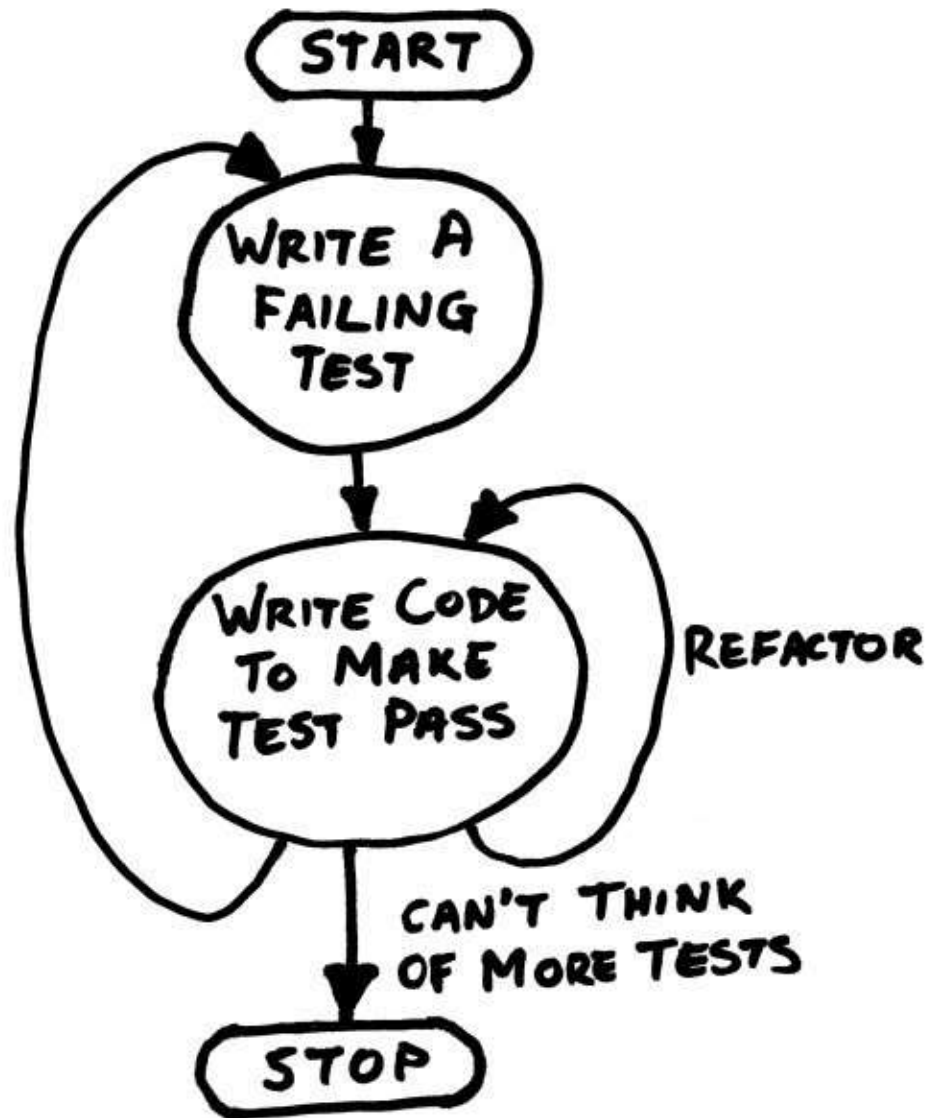
- ▶ *Model Driven Development* (MDD) is a paradigm for writing and implementing computer programs **quickly, effectively and at minimum cost**
 - ▶ MDD is an approach to software development where **extensive models are created before source code is written**
 - ▶ A primary example of MDD is the Object Management Group (OMG)'s Model Driven Architecture (MDA) standard
- 

Agile MDD – Iteration Modeling

▶ Thinking Through What You'll Do This Iteration



Test Driven Development



Myths

▶ Clients

- A general description of the objectives is sufficient to begin writing program
- Requirements are constantly changing, but the software is flexible and can easily adapt

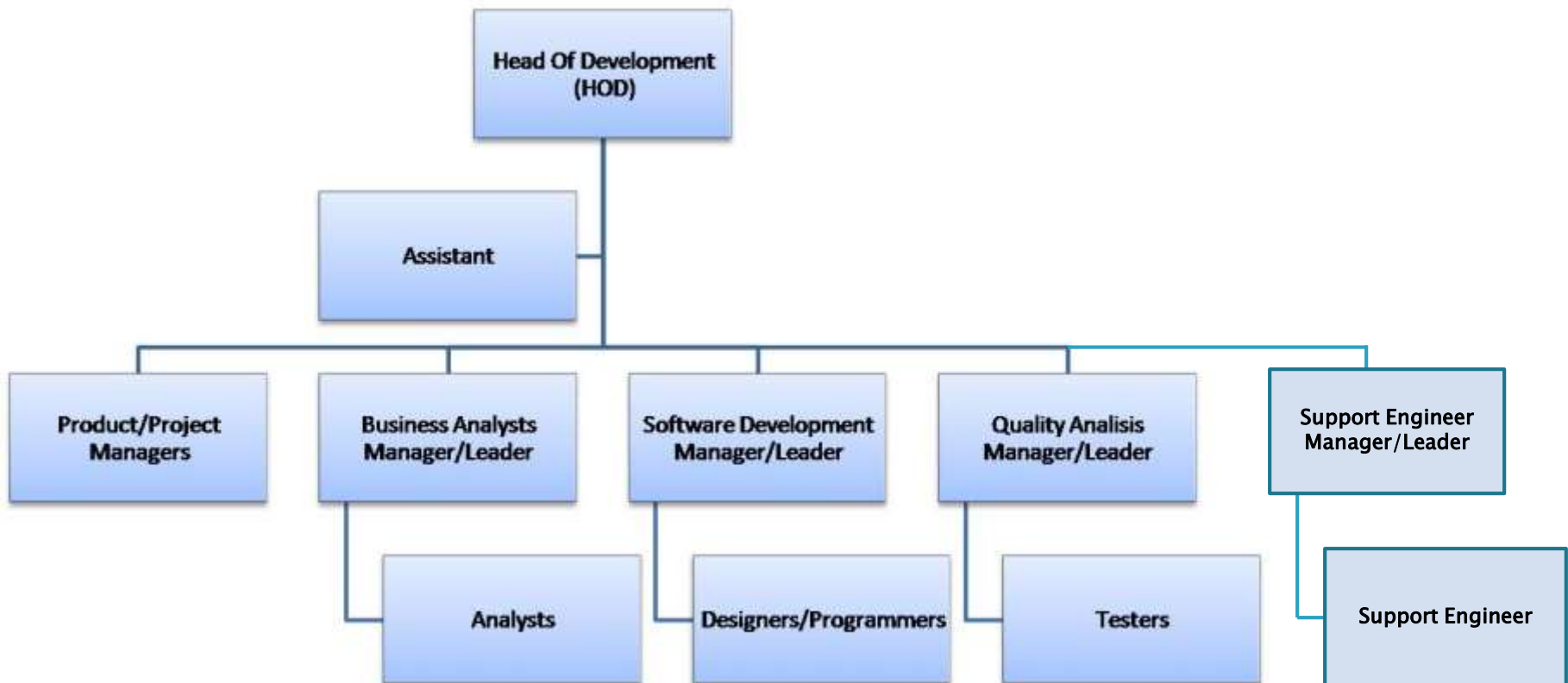
▶ Developers

- Once the program is written and it is functional, our role has ended
- Until the program doesn't work, we can not assess the quality
- The only good product is the functional program
- Software Engineering will create voluminous and unnecessary documentation and will cause delays

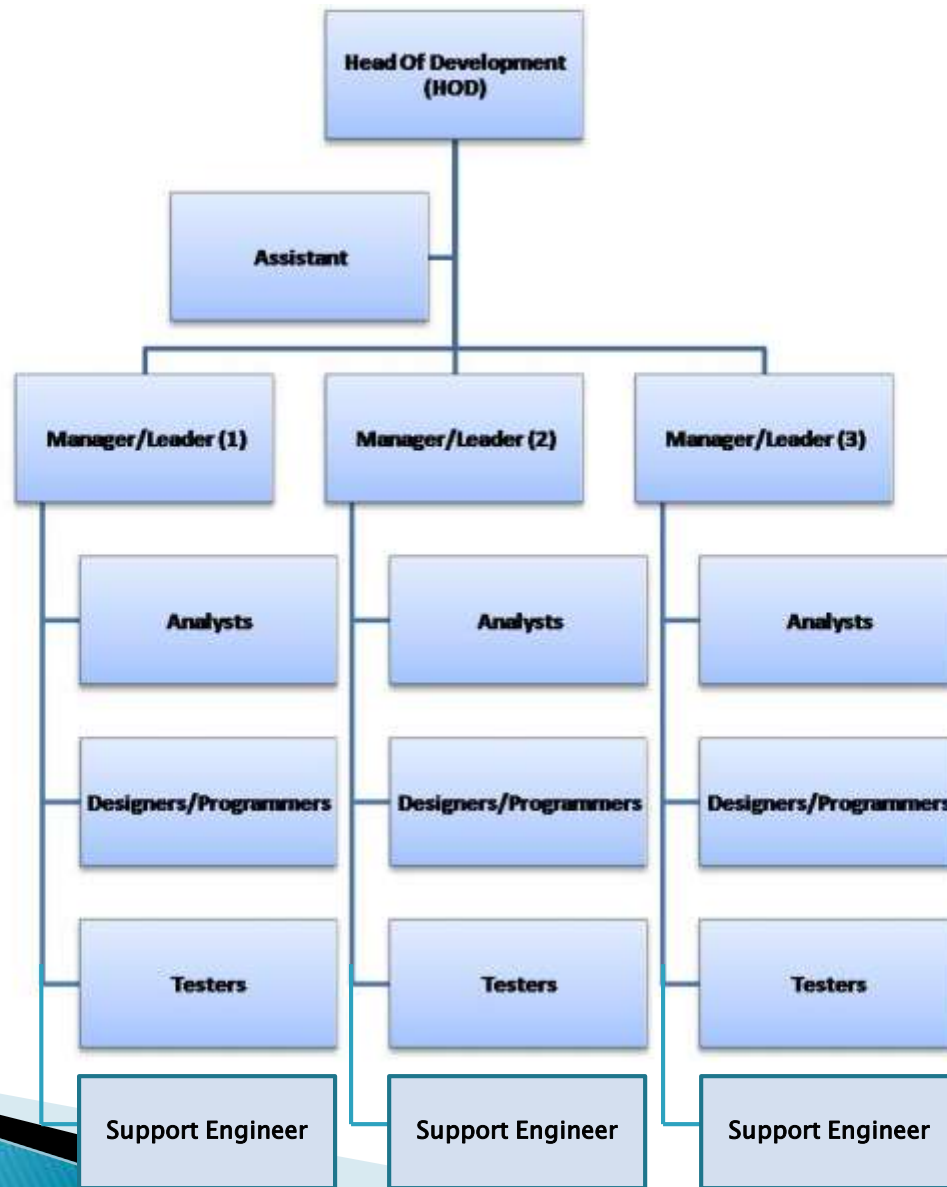
How will we do?

- ▶ Very good communication with the customer, which is part of the team (SCRUM)
- ▶ After each stage you get a finished product, which usually can not be restored during the next steps (Waterfall)
- ▶ As a member of the "Team", I will support you as much as possible (LEAN)
- ▶ Everyone will be encouraged to do what he likes more (XP)
- ▶ I DO NOT come with continuous changes in my requirements (NOT AGILE)
- ▶ We do NOT do a risk study (NOT SPIRAL)

The hierarchy of a software company – Departments



The hierarchy of a software company – Projects



Software Development Life Cycle

- ▶ Requirements engineering
- ▶ Architectural design
- ▶ Detailed design
- ▶ Implementation
- ▶ Integration
- ▶ Validation
- ▶ Verification
- ▶ Deployment
- ▶ Maintenance

How does a project start?

- ▶ A customer wants to
 - Improve productivity
 - Solve a personal problem
 - Advertise the products they sell
 - Easily manage branches in the country
- ▶ An interesting project
- ▶ One idea, the need to manage my daily expenses, etc.
- ▶ From this point onward follow the Requirements Engineering!

Requirements Engineering – Definition

- ▶ Process of understanding customer needs and expectations regarding our app
- ▶ A well-defined stage in the Software Development Life Cycle
- ▶ Which functionalities do we expect one application to have
- ▶ How should the system behave and what are its characteristics
- ▶ Also known as *Software Requirements*, *Software Requirements Specification*, *Specification*

Requirements Engineering – Example

- ▶ Create a C ++ program to perform the sum of two matrices read from the file.
- ▶ +:
 - We know the programming language
 - We know that reading is done from file
- ▶ -:
 - We do not know what to do with two matrices which do not have the same dimensions
 - What about the result?

Requirements Engineering – Who?

- ▶ Due to the multitude of types of interactions that may exist between users, business processes, hardware, etc. there may be different types of requirements, from simple applications to complex applications
- ▶ Requirements analysis process involves choosing, documenting these types of requirements and constructing documents that will be the basis for the building system

▶ **Who's in charge?** Project Manager, Program Manager or Business Analyst

Requirements Engineering – Why?

- ▶ Studies show that poor attention given to requirements analysis is **the most popular cause of vulnerabilities in projects**
- ▶ Many organizations have spent so much money on software projects that ultimately **were not what was originally wanted from them**
- ▶ At this moment many companies are investing time and money to make **effective requirements engineering**

Requirements Engineering – Steps

1. Determination of boundaries for application
2. Finding the client
3. Identification of requirements
4. Requirements analysis process
5. Requirements specification
6. Requirements management

Case study: scholarships

RE – Determination of boundaries for application (1)

- ▶ As a first step, it aims to identify **how this new application will be integrated in the environment** that will be designed
- ▶ What will the **purpose of the application** be?
- ▶ What will the **limits of application** be?

RE – Finding the client (2)

- ▶ The objective of recent years: Who is the **end user** (customer) using the actual application?
- ▶ As a result, we will know exactly **what people will be directly or indirectly affected** by the implementation of this product
- ▶ We will know **whom to ask for any clarifications**

RE – Identification of requirements (3)

- ▶ The requirements are collected from several groups which have been identified in the previous step
- ▶ Identify what they want the application to perform
- ▶ The depth depends on:
 - The number and size of the groups
 - The complexities of the business process
 - The size of the application
- ▶ Problems encountered at this stage
 - Ambiguity in understanding the processes
 - Inconsistency in understanding the same process
 - Insufficient data
 - Changes in requirements after the project started

RE – Identification of requirements – Who? (3)

- ▶ This person must interact directly with several working groups
- ▶ It has to do with conflicting ideas
- ▶ Must have communication skills and must work with people
- ▶ Must have knowledge of programming
- ▶ Finally we need to agree on customer requirements

RE – Identification of requirements – Methods (3)

- ▶ Interviews with future users and user groups
- ▶ Using existing documentation (manuals, organizational charts, system specifications, etc.)
- ▶ Methods:
 - Prototypes
 - "Use case" diagrams
 - Data flow and processes diagrams
 - User interfaces

RE – Requirements analysis process (4)

- ▶ It makes a structured analysis using specific techniques:
 - "Animation" requirements
 - Reasoning
 - Critical look in terms of knowledge
 - Consistency Checking
 - Analogical and based on examples reasoning

RE – Requirements specification (5) 1

- ▶ It is done in a **clear** and **unambiguous** manner
- ▶ **Writing a document** which specifies the requirements is mandatory!
- ▶ This document will circulate to all those involved in this phase: customer, user groups, development and testing teams
- ▶ The document will be used:
 - For validating requirements by clients
 - For the contract between the client and the development team
 - For the design of the system basis, created by developers
 - For the planning basis
 - As source for making test scenarios

RE – Requirements specification (5) 2

- ▶ Must capture the **customer's view about this product**
- ▶ It is the result of **collaboration between the user** (who is not an expert) and **system analyst** (which captures the situation in technical terms)
- ▶ It is possible that the specifications of the requirements be made in two separate documents:
 - **User requirements** – written clearly using use cases (for the user)
 - **System requirements** – described using a mathematical model or programmatically (for developers and testers)

RE – Requirements specification (5) 3

- ▶ In **user requirements** there should not occur technical concepts (communication protocol, MD5 encryption, HTTP, IP, etc.)
- ▶ In **system requirements** should appear export data format (Json, XML), the server address from which it is perform the reading and the place where the log files are stored

RE – Requirements specification (5) 4

- ▶ The level of detail:
 - **Low** – can be vague (does not help during development and testing)
 - **High** – a lot of work, sometimes useless (more precisely and more clearly)
 - ▶ Example:
 - Create a program to make the sum of two matrices.
 - Create a C # program that has class attributes Matrix n, m integer representing the number of rows and columns and type int array [3] [3] representing the matrix elements. Matrix class methods are available
-

RE – Requirements specification (5) 5

- ▶ Types of requirements:
 - **User requirements:** about where the system will be used, efficiency, the lifetime of the product (*the product will be used by the financial department*)
 - **Functional requirements:** how to perform certain calculations, how to manipulate data (*salary tax is 16%*)
 - **Performance requirements:** how certain functions are called quantitative, qualitative (*the system allows 1000 queries per second*)
 - **Constraints:** it *will not allow two people to simultaneously enter data into tables*

RE – Requirements specification (5) 6

- ▶ It is an ongoing process that captures all aspects of identifying requirements and additionally provides verification, validation
- ▶ To be useful, it must ensure unambiguous requirements, errors elimination and omissions completion

Use cases 1

- ▶ They use **actors** (elements which interact with the program):
 - Human users
 - Software elements (ie software that processes information gathered from the Internet)
 - Hardware (ie barcode reader, mobile phones, etc.)
- ▶ **Use cases**
 - They describe how the actor interacts with the system
 - How the system reacts after these actions
 - What is the visible result for actors

Use cases 2

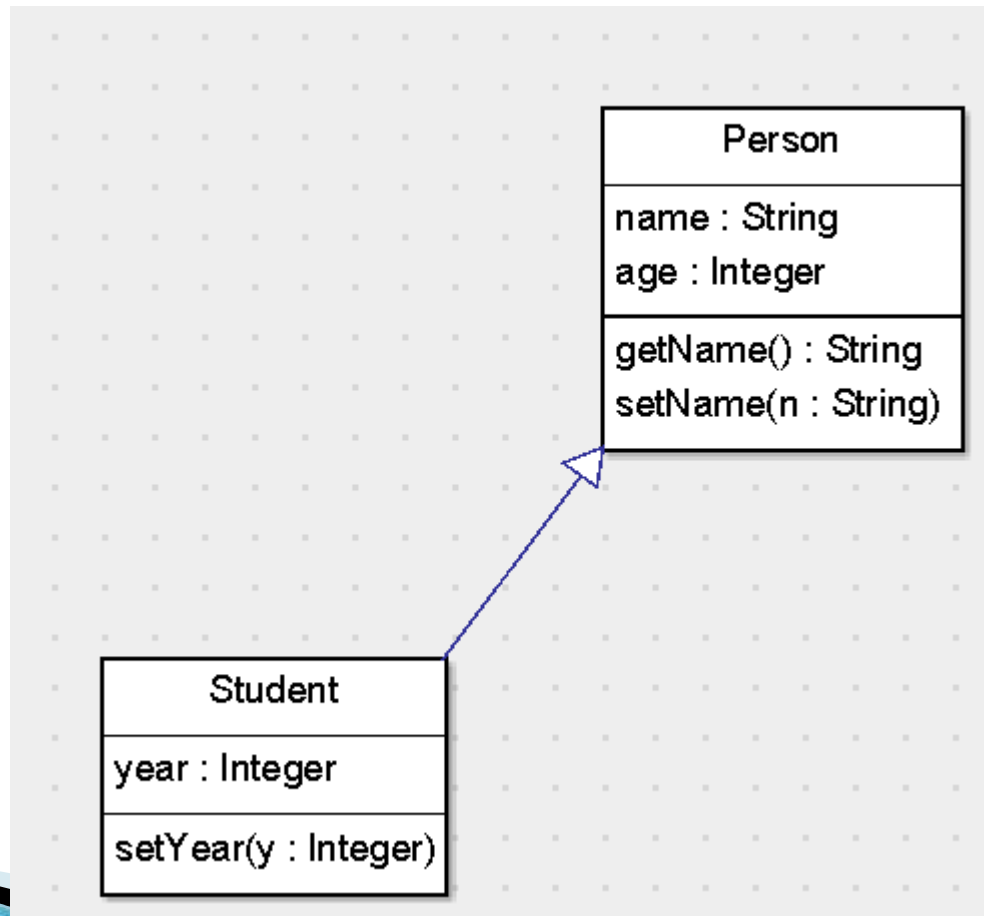
- ▶ What they do not contain:
 - Class diagrams
 - The modular structure of the program
 - Type of input and output data
- ▶ Use Case – Type of content:
 - **In short** – describes the main success story
 - **Casual** – contains what should be done in case something happens
 - **Detail** – it presents in detail all possible situations

Use Case – Example

- ▶ **In short:** The program should be able to add 2 matrices
- ▶ **Casual:** The program should be able to add 2 matrices, if they have the same number of rows and columns, otherwise it will display an appropriate error message
- ▶ **Detailed:** The program must be able to add two matrices of integers, read from the keyboard, if they have the same number of rows and columns, and the resulting matrix is displayed in a file "rezultat.txt" one line at a time. Otherwise it will display an appropriate error message to a file "mesaj.txt" in the current directory.
(Should it be added something more?)

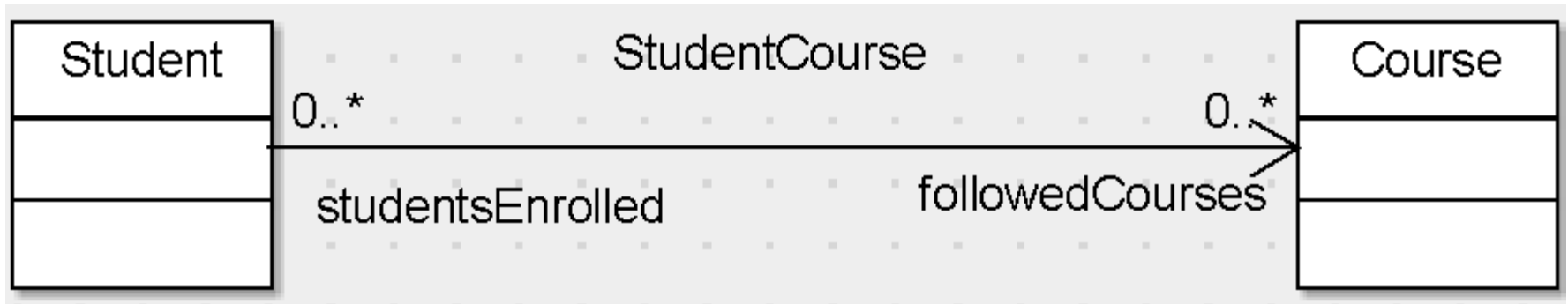
UML – Class diagrams

- ▶ Inheritance relation (generalization)



Relations of association 1

- ▶ Relation Student – Course
 - **Student**: follow 0 or more courses, I **know** what courses I have
 - **Course**: can be followed by more students, I **don't know** which are the students

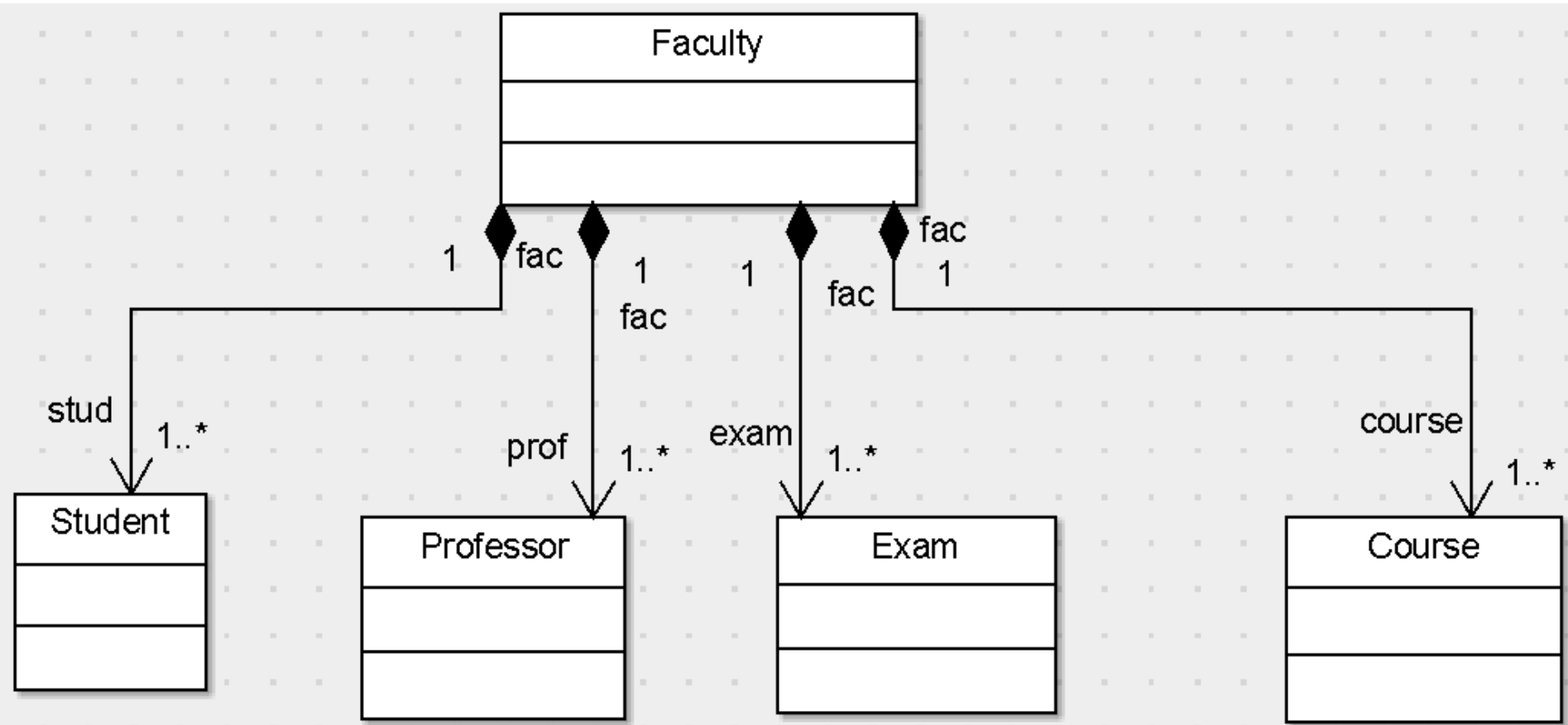


Relations of association 2

- ▶ Relation Disciplină – Professor
 - **Course:** are kept by a professor, whom I know
 - **Professor:** I can teach more courses, I know the courses I teach



“hasA” relation (composition)



ArgoUML

- ▶ Link: <http://argouml-downloads.tigris.org/argouml-0.34/>
- ▶ “zip” version must be unpacked
- ▶ Requires Java
 - In Path add c:\Program Files\Java\jdk1.6.0_03\bin
 - Variable
JAVA_HOME=c:\Program Files\Java\jdk1.6.0_03\

Conclusions

- ▶ Deployment models
- ▶ Requirements engineering
- ▶ OOP

- ▶ Suggestions:
 - Work at the lab with ArgoUML (which allows you to do forward and reverse engineering)
 - Check in advance the topic of laboratory and practice at home in advance the introductory steps (installation, finding options, implementing a very simple example ...)

Bibliography

- ▶ Anil Hemrajani, Agile Java Development with Spring, Hibernate and Eclipse, 2006
- ▶ Dorel Lucanu, OOP Principles

Links

- ▶ XP: <http://www.extremeprogramming.org/rules.html>
- ▶ Agile: <http://agilemanifesto.org/>
- ▶ Scrum: <http://jeffsutherland.com/oopsla/schwapub.pdf>
- ▶ Lean: http://www.projectperfect.com.au/info_lean_development.php,
http://dtic.mil/ndia/2006cmmi/wednesday/3C7_Card.pdf
- ▶ V-model: <http://en.wikipedia.org/wiki/V-Model>,
http://en.wikipedia.org/wiki/V-Model_%28software_development%29
- ▶ Project Management White Paper Index:
http://www.projectperfect.com.au/wp_index.php
- ▶ Requirements analysis process:
<http://www.outsource2india.com/software/RequirementAnalysis.asp>
- ▶ ImageCup 2009:
<http://fiistudent.wordpress.com/2008/12/10/imagine-cup-2009-ce-ar-fi-daca-intr-o-zi-am-ajunge-toti-la-muzeu/>
- ▶ Course 2 PE – Ovidiu Gheorghieș:
<http://www.infoiasi.ro/~ogh/files/ip/curs-02.pdf>
- ▶ Software house: http://en.wikipedia.org/wiki/Software_house